Request for Proposal: Annotation Platform Development

Executive Summary

This Request for Proposal (RFP) seeks qualified vendors to develop a comprehensive annotation platform for linguistic and media content. The platform will support both public and private deployment models, with robust features for annotation, collaboration, and content management.

Project Overview

The proposed system will be a web-based platform with optional desktop client, designed to handle linguistic annotations, media alignment, and collaborative editing. The platform must support both public-facing deployments (similar to Zenodo.org) and private, permissioned deployments for sensitive or pre-publication content.

Submission Requirements

1. **Proposal Format**
    - Maximum 20 pages, excluding appendices
    - PDF format
    - 12pt font, 1-inch margins
2. **Required Sections**
    - Company Profile
    - Technical Approach
    - Project Timeline
    - Cost Breakdown
    - Team Qualifications
    - References
    - Implementation Plan
    - Support & Maintenance Plan
3. **Timeline**
    - RFP Release Date: June 1, 2025
    - Questions Due: June 15, 2025 (2 weeks for initial questions)
    - Proposal Due: July 15, 2025 (1 month for proposal preparation)
    - Vendor Selection: August 1, 2025 (2 weeks for evaluation)
    - Project Start: September 1, 2025 (1 month for contract negotiation and setup)
4. **Evaluation Criteria**
    - Technical Solution (40%)
    - Cost (20%)
    - Implementation Timeline (15%)

- Team Experience (15%)
- Support & Maintenance (10%)

Budget

Vendors are requested to provide a detailed budget proposal that includes:

- Development costs
- Testing and QA
- Documentation
- Training
- One year of support and maintenance
- Optional: Ongoing maintenance costs

Please provide a total project budget and break down costs by major components. Include any assumptions or conditions that affect the proposed budget.

Terms and Conditions

1. **Intellectual Property**

   - All deliverables, including but not limited to source code, documentation, and any custom-developed software, shall become the exclusive property of Kavon Hooshiar upon final payment
   - The vendor must transfer all rights, title, and interest in the deliverables, including all intellectual property rights
   - Source code must be delivered with:
     - Full, unrestricted ownership rights
     - All necessary licenses and permissions
     - No third-party encumbrances or restrictions
   - Documentation must be provided in editable format
   - The vendor must warrant that they have the right to transfer all intellectual property rights
   - The vendor must indemnify against any claims of intellectual property infringement

2. **Warranty**

   - 12-month warranty period for all deliverables
   - Bug fixes and critical updates included during warranty
   - Performance guarantees must be specified

3. **Payment Terms**

   - 30% upon contract signing
   - 40% upon completion of core functionality
   - 30% upon final acceptance
   - Payment terms: Net 30

4. **Confidentiality & Licensing**

   - Proposals will be treated as confidential during the evaluation process

  - The final developed software should be suitable for educational use
  - While the source code will be owned by Kavon Hooshiar, the intention is to make the software available for educational purposes
  - Vendors should indicate their experience with:
    - Open source development
    - Educational software projects
    - Documentation for educational users
  - Source code and documentation should be developed with educational use in mind

Vendor Requirements

1. **Company Qualifications**
   - Minimum 5 years in software development
   - Experience with similar projects
   - Financial stability
   - Team size and composition
2. **Technical Capabilities**
   - Web development expertise
   - Desktop application development
   - Git integration experience
   - Media handling capabilities
   - Security implementation experience
3. **Support Requirements**
   - 24/7 support availability
   - Response time SLAs
   - Bug fix turnaround times
   - Update deployment process

Selection Process

1. **Evaluation Timeline**
   - Initial review: 1 week
   - Technical interviews: 1 week
   - Reference checks: 3-5 business days
   - Final selection: August 1, 2025
2. **Contact Information**
   - Primary Contact: Kavon Hooshiar
   - Email: kavon.hooshiar@gmail.com

Technical Requirements

[The existing requirements document follows...]

Project Requirements Document

This document combines all sections of the project requirements in a single file.

Table of Contents

---

1. Modes of Operation

This section defines the deployment-level behavior of the platform and establishes foundational access rules, which dictate all subsequent UI, permission, and document visibility logic.

1.1 Public vs Non-Public Mode

Deployment-Time Setting

- The mode (`public` or `non-public`) is **hardcoded** during deployment and cannot be toggled from the admin dashboard.
- Switching modes post-deployment would require migration of data to a new instance, including exporting/importing collections and possibly restructuring Git repositories.

Public Mode

- Modeled after platforms like Zenodo or Archive.org.
- **Publishing model**: Documents pass through pre-publication stages and, once published, become **fully public** and discoverable.
- **User authentication**:
  - Not required to browse or view public documents.
  - Required for personalization features (favorites, saved searches, personal tags, resume state, etc.).
- **Groups/collections**:
  - Optional but available to facilitate collaborative editing and curation.

- **Commenting**:
    - Publicly visible annotations may allow commenting by authenticated users, depending on the document's settings.

Non-Public Mode

- Used as a secure work environment before public release.
- **Access is entirely permission-based**:
    - All documents must be part of a **group (collection)**.
    - **Only users with explicit roles in a group can access its documents.**
- Sharing is enabled via permissioned invites:
    - A link can be sent to invite someone to view or collaborate on a document within a group.
    - Email-based invite flow allows pre-approval before account creation.
- **Publishing** in this mode means marking a document as final or reviewed within the group context; documents remain private within the instance.

Behavioral Differences

Feature	Public Mode	Non-Public Mode
Document visibility	Public after publish	Always restricted
User registration required	No (optional for browsing)	Yes
Collections	Optional	Required for all documents
Sharing by link	Public link for published	Authenticated, invite-only access
Git repo organization	One global repo	One repo per collection

2. Data Structure & File Architecture

2.1 Git-based Storage

- **Canonical storage model** is Git:
    - Each **collection** is a Git repository.
    - Repositories may be:
        - Hosted internally by the app's own Git server (recommended for non-public config).
        - Externally hosted (e.g., GitHub, GitLab) in public config or by user preference.
- Repositories must support **Git LFS (Large File Storage)** for media assets such as `.mp3`, `.wav`, `.mp4`, and `.mov`.
- Git storage enables:
    - Precise versioning of all files including annotation JSON and manifests.
    - Support for both real-time editing sessions (collapsing into commits) and manual pushes from external tools.
    - Flexible collaboration and diffs, reverting, merging.

2.2 File Structure

- Files are organized into **3 primary logical levels**:

  1. **Collection** — the repository root.
  2. **Document Group** — top-level folders within the repo (e.g., `bundle-001/`).
  3. **Document Files** — the contents within a document group (e.g., media, annotations, notes, PDFs).

- Folder structure is user-defined, but constrained by a **nesting level setting**:

  - Default: `1` (only the top-level folders are treated as logical units).
  - Maximum: `10`.
  - Deeper folders beyond this threshold are flattened logically (still shown in UI, but grouped under the last displayed parent).

- The term used to describe "document groups" in the UI (e.g., "bundles", "items", "recordings") is configurable per app instance.

2.3 Metadata & Manifest

- Every logical folder (document group or nested group) includes a `manifest.json` file.

  - Required fields:
    - `id`: stable identifier, often matching folder name.
    - `title`
    - `languages` (can be one or more)
    - `media_types`
    - `tags`
    - `parent_path`: string path of parent folder, used to reconstruct hierarchy even if moved.
    - `inferred`: boolean flag indicating if values were auto-generated.
    - Any additional `custom_fields`.

- Top-level `collection-index.json`:

  - Summarizes all `manifest.json` entries.
  - Fields indexed:
    - `id`
    - `title`
    - `languages`
    - `media_type`
    - `tag_list`
    - `parent_path`
  - Git-tracked like any other file.
  - Auto-updated:
    - On document-level save
    - On manifest regeneration
    - Periodically via scheduled task

2.4 Manifest Auto-generation

- Manifests are generated automatically:

    - On first import to app
    - If a folder lacks a manifest

- User is notified of this during import.

- File types are all included by default except for system files (e.g., `.DS_Store`, `.gitignore`, etc.).

- Each collection can define extensions to ignore via UI.

- When auto-generating manifests:

    - File types (e.g., `.mp3`, `.json`, `.pdf`) are classified into roles (e.g., "media", "annotation").
    - File metadata and structure are used to infer media types, alignment modes, etc.
    - Document titles and language data may be blank until entered manually.
    - Any inferred data is marked explicitly in the manifest.

2.5 Manifest Commit Behavior

- Manifest generation behaves as a normal **edit action**:

    - The files are staged for commit in the Git repo.
    - Final commit occurs when user explicitly saves or publishes the document.
    - This allows reviewing auto-inferred metadata before it becomes part of version history.

3. Annotation Data Model

3.1 Native Format: JSON

The system stores annotations in a structured, extendable JSON format. Each document's annotation data is stored in a `.json` file that conforms to a defined schema, supporting:

- **Tiers**: Logical groupings of annotations, e.g., original, translation, morphological gloss, etc.
- **Annotations**: Individual entries on a tier, each aligned to either a time range or a logical index (e.g., line number).
- **Spans**: Optional inline subdivisions of an annotation's text, allowing partial markup/tagging (e.g., tagging only one word in a sentence).
- **Tags**: References to user-defined concepts (e.g., linguistic features), which can be applied to:
    - Entire annotations
    - Inline spans
    - Document-level or tier-level metadata

JSON Schema Elements

Each annotation file includes:

- `tiers[]`: Array of tiers

    - `id`: unique ID
    - `name`
    - `type` (e.g., original, translation)

    - `lang`
    - `direction` (`ltr` / `rtl`)
    - `parent`: optional tier ID it depends on
    - `annotations[]`:
        - `id`
        - `start` / `end`: timecodes or logical IDs
        - `text`
        - `spans[]`: optional array
            - `start_char` / `end_char`
            - `tag_ids[]`
        - `tags[]`: tag IDs applying to the full annotation

- `tags[]`: tag definitions (or links to shared tagset)

- `metadata`: optional key-value data for the file

3.2 Tier Relationships

- A tier may depend on a parent tier (e.g. word-level dependent on sentence-level).
- In time-aligned mode, dependent annotations are constrained within the parent's timecodes.
- In logical alignment, parent annotations serve as the primary units (e.g., lines or sentences) and child annotations are linked via IDs.

This relationship is maintained internally but also mapped faithfully to/from ELAN.

3.3 Inline Tag Support

Tags may be associated not only with full annotations but also with **spans of text within annotations**.

- Displayed as stylized inline text (e.g., underlined or color coded).
- Hover/click shows tag details.
- Clicking a tagged span activates the annotation and shows tag info.

Internally:

```
{
  "text": "She gave the book",
  "spans": [
    {
      "start_char": 4,
      "end_char": 8,
      "tag_ids": ["verb"]
    }
  ]
}
```

Exported to ELAN or similar:

```
<annotation>
  She <span tag="verb">gave</span> the book
</annotation>
```

3.4 Tagsets

- Tags are globally defined per collection and referenced by ID within annotation files.
- Tags can include:
  - `id`, `label`, `description`, `color`, `category`
  - User-defined metadata fields
- Private tags:
  - Only visible to the creator
  - Stored separately in user metadata but linked to annotation via tag ID

3.5 Constraints

The annotation schema enforces:

- Non-overlapping annotations on the same tier
- Spans must be fully within the text length
- Tags must be declared in the collection tagset
- Unique IDs for annotations and tiers
- Time-alignment vs. logical alignment must be consistent within each file

---

3.6 Compatibility with ELAN

- Import:
  - ELAN tier dependencies, IDs, and annotations are preserved
  - ELAN annotations with `<span>`-style markup parsed into spans
- Export:
  - Native annotations exported to `.eaf` with:
    - Tier hierarchy
    - Timecodes/logical IDs
    - Inline tags embedded as `<span>`-style text

Section 4: Viewer UI Requirements

4.1 Layout

- Responsive layout adjusts for device screen size and orientation.
- For video:

  - Landscape: video plays beside the text display.
  - Portrait: video appears above or below the text.
- For audio:
  - Scrollable audio bar appears above or below the text.
- Text layout modes:
  - **Inline mode**: Selected tiers are shown in a single vertical flow per timestamp/logical unit.
  - **Page (book-style) mode**: Up to 2 pages on phones, up to 4 on large screens. Each page can show one or more tiers.
- Layout adapts to available space and can be toggled manually.

4.2 Media Interaction

- Media playback:
  - Automatically scrolls to the currently active annotation.
  - Clicking an annotation seeks media to its start time and pauses playback.
  - Users can loop playback of an annotation.
- User controls:
  - Keyboard shortcuts (e.g., spacebar to toggle loop, arrow keys to seek).
  - Touch gestures (e.g., double-tap to loop).
  - Toggle for "seek-on-click" to either pause or resume playback.

4.3 Tier Display & Controls

- Users can:
  - Show/hide individual tiers.
  - Reorder tiers interactively.
  - Zoom in/out on text (via font scaling), affecting wrapping and layout density.
- Small-screen view:
  - Option to switch to a one-item-at-a-time view (logical annotation per page).
  - Still able to switch back to full-page layout.

4.4 Inline Tag Interaction

- Inline tags:
  - Always rendered visibly inside annotation text.
  - On hover or tap, tag metadata appears in a tooltip or bubble overlay.
- Clicking a tagged word both:
  - Selects the full annotation, and
  - Opens the tag metadata bubble.
- On larger screens:
  - A floating or docked panel shows all tags for the current annotation in focus.

4.5 Fallback States

- No media file:
  - Media player UI is hidden.
- Single tier:
  - Page auto-falls back to inline layout with just that tier.

    - If previously in page mode with multiple pages, fallback applies automatically.
- No alignment info:
    - App assumes default ordering from document file.
    - Alignment fallback is implicit via logical annotation IDs or order.

Section 5. Content Management (CMS)

This section defines how documents and data are ingested, edited, validated, and managed in the system. It covers supported formats, editing tools, role-based permissions, and document settings.

5.1 Supported Uploads

Supported formats include:

- **Media**: `.mp3`, `.wav`, `.mp4`, `.mov`
- **Annotation files**:
    - Native JSON schema
    - ELAN `.eaf`
    - `.txt`, `.csv`, `.docx`, `.pdf` (converted to JSON with user-guided tier suggestions)

Import behavior:

- For `.txt`, `.docx`, or `.pdf`, the system:
    - Segments by line and/or sentence
    - Suggests annotations and tier assignments
    - Prompts user to validate or revise tiering
- All uploaded files (except system files) are logged and included in folder manifests
- Unknown file types can be flagged or ignored based on collection config

---

5.2 Document Creation

Options:

- Upload-based creation: user uploads annotation/media files
- UI-based creation: user creates a blank document and builds tiers manually
- Document creation triggers inference engine for:
    - Languages (from tier labels or content)
    - Media type
    - Tier relationships
- Required metadata (title, languages, alignment mode) flagged if missing, but not blocking until publish

---

5.3 Editing Tools

All editing happens via a UI linked to Git-backed versioning.

Supported editor functions:

- Edit annotation text and metadata
- Adjust timecodes (if media is present)
- Add/remove annotations
- Add/remove tiers
- Reorder annotations (drag/drop)
- Merge or split annotations
- Inline tagging of spans (with metadata)
- Change tier direction (ltr/rtl)
- Define parent/child tier constraints

UI feedback:

- Inline warnings for overlapping timestamps, orphan annotations, or bad tier references
- Tooltip access to tag metadata and span information
- Live preview of timecode interactions

5.4 Roles

The system has four access levels:

- **User**
  - Can view public documents
  - Can bookmark annotations
  - Can create personal tags and notes
- **Contributor**
  - Can create and edit their own documents
  - Can import, tag, annotate, and publish their own content
- **Moderator**
  - Can be assigned edit or review access to any document
  - Can approve submissions for publication
  - Can manage permissions within a group
- **Admin**
  - Full access to all collections and data
  - Can edit any document, create tagset templates, and set instance-wide configs

Role-based visibility and permissions are scoped either to the entire instance (in public mode) or to collections (in non-public mode).

5.5 Document-Level Settings

Each document has an editable metadata panel that includes:

- **Basic Info**
  - Title (required)
  - Author(s)
  - Languages involved
- **Media Alignment**
  - Automatically set to time-aligned if media is present
  - Logical alignment used otherwise (e.g. sentence order)
- **View Mode**
  - Default UI rendering: inline or book-style
- **Tagset**
  - Choose from a predefined tagset or define a new one
- **Visibility & Access**
  - Controlled via group membership in non-public mode
  - Public mode uses publishing status (draft, under review, published)

Section 6: Query & Discovery

6.1 Filters & Query Interface

The application supports rich filtering and search capabilities:

- **Tag-based filters**: Select one or more tags to find relevant annotations.
- **Tier-based filters**: Filter by tier language or type (e.g., "translation", "morpheme").
- **User-specific tags**: Logged-in users can filter by their own private tags.
- **Media-based filters**: Filter results by presence of audio or video media.
- **Hybrid full-text + tag search**: Users can perform combined searches (e.g., "find all translations containing 'run' tagged as verb").

6.2 Search Results Rendering

Matching annotations should be rendered in a rich, context-preserving list:

- Each result shows the annotation, tier name, and surrounding annotations as context.
- If media is available, include an inline video/audio player scoped to the annotation's time span.
- Results are interactive: users can click to jump to that annotation in full viewer.
- Book-style context should not be required for results display (they are treated as standalone viewer slices).

6.3 Saved Searches & Tag Collections

Users can create, label, and manage saved queries:

- **Saved search** = persistent tag+text filter (like a playlist or smart folder).
- Users can choose **public** or **private** visibility.
- Saved searches are accessible via:
  - Sidebar
  - Dashboard
  - "My Library" view

6.4 Use Cases Supported

- Linguists exploring specific constructions (e.g. tag: applicative)
- Learners collecting verbs with a specific conjugation pattern
- Editors reviewing all annotations tagged "needs attention"

Section 7: Versioning & Collaboration

7.1 Real-Time Editing Model

- **Editing sessions** operate like collaborative Google Docs sessions:
  - Changes from all users in a session are visible live.
  - Changes are temporarily cached until a save/commit action.
- **Commit behavior:**
  - One save = one Git commit.
  - Changes from a session are bundled as a single commit.
- **Session joining:**
  - Users from the same group/collection can join the same session.
  - UI discourages multiple simultaneous independent sessions to minimize merge conflicts.

7.2 Git-Based Storage and Versioning

- Each document or collection exists in a **Git repository** (internal or external).
- All document versions are:
  - **Git-tracked** — every save operation commits a new version.
  - **Diff-able** — diffs are viewable via UI between any two states.
  - **Restorable** — previous versions can be restored by reverting the Git state.
- Git LFS (Large File Storage) is used to manage large audio/video media.
- In non-public configs:
  - Each **collection** = one Git repo.
  - Git server may be internal or connect to GitHub/GitLab.

7.3 Merge Conflict Handling

- If multiple sessions or offline edits conflict:
  - Merge conflict resolution UI is triggered.
  - Users can select which version to keep or manually merge.

7.4 Version Publishing

- A **published version** is frozen in Git and tagged.

- Future edits fork a new working version (new Git commit thread).
- Published versions are:
  - Citable (e.g., by UUID or configured DOI-like slug).
  - Immutable (can't be edited without forking).

7.5 Collaboration & Review Tools

- Commenting:
  - On whole documents, tiers, annotations, or spans.
  - Threaded, with replies and resolution markers.
- Review Workflow:
  - Documents can be assigned to **moderators** for review.
  - Document states include:
    - Draft
    - In Review
    - Rejected
    - Approved / Published
- Comments remain visible for all collaborators.
- Comments can be marked **resolved** once addressed.

Section 8: Import & Export

8.1 — Import Support

The system supports importing documents and data from a variety of sources:

Supported Import Formats

- **ELAN `.eaf` files**: including tier types, annotations, media alignment, and dependencies.
- **Native JSON format** (our app's schema).
- **Plain text, `.docx`, `.pdf`**: parsed into annotations using rules like "line per annotation" or "sentence per annotation".
- **CSV/TSV**: structured as rows with timestamps, text, tier labels, etc.

Behavior During Import

- ELAN imports preserve:
  - Tier types and constraints (e.g., hierarchical tier dependencies).
  - Original ELAN IDs for traceability and round-tripping.
  - Both symbolic and time-aligned tiers.
- Inline tags in ELAN are detected via `<span ...>` style syntax in annotation text and stored natively as span metadata.
- When importing plain text or documents, the system prompts the user to:
  - Select how to segment the text (lines, sentences, paragraphs).
  - Assign content to a tier (e.g., original, translation, gloss).
- Metadata such as title, language, and tier names are:

  - Partially inferred (e.g., from filenames or ELAN fields).
  - Prompted for completion before saving.
  - Fields that are required (e.g., document title, languages) are validated and flagged prior to publishing.
- Tier types on import:
  - Mapped to the app's native tier taxonomy.
  - Users can configure collection-specific tier types and override mappings from ELAN when needed.

8.2 — Export Support

The system supports rich export options for interoperability and archival:

Supported Export Formats

- **Native JSON**
- **ELAN `.eaf`**
- **CSV or TSV**
- **Text-only (e.g., original and translation only)**
- **PDF (book-style layout)** with:
  - Optional inclusion of media snippets for digital use
  - Printable version with media stripped

Export Options

- Select which tiers to include.
- Choose to include or omit inline tags.
- ELAN `.eaf` export:
  - Required to contain timecodes; if missing, synthetic timecodes are created with user warning.
  - Original ELAN IDs maintained if document was imported from ELAN.
  - Nested span annotations are flattened as `<span ...>` markup.
- Timecode vs logical alignment:
  - Timecodes always preferred when media is present.
  - For logical-only documents, export with sequence-based alignment unless exporting to ELAN (which requires timecodes).
  - Export to ELAN without media prompts the user to confirm auto-generated timestamps.
- Export filters:
  - "Export only filtered/tagged annotations" is not supported at this stage.
- PDF exports:
  - Format mirrors the in-app book-style layout.
  - Users configure which tiers show on which "pages".

  - Optional: generate interactive PDFs with embedded audio snippets.

Section 9. Personalization

9.1 Resume & Favorites

- **Resume State Tracking**
  - Each user's viewing progress is automatically tracked per document.
  - On returning to a document, the user resumes from the last viewed annotation.
  - This behavior is enabled by default.
- **Favorites**
  - Users can mark entire documents as favorites.
  - Annotations can be bookmarked individually for quick access.
  - Favorite status is stored in the user's personal account.
  - These features are only available to logged-in users.

9.2 Private Tags & Notes

- **Personal Tags**
  - Users can create tags that are private to their account.
  - These tags:
    - Are visible only to the user who created them.
    - Are styled differently (e.g., dimmed, icon-badged) to distinguish from global tags.
    - Can be used in personal search queries (e.g., "my tags: verb").
    - Can apply to entire annotations or inline spans.
- **Notes**
  - Free-text notes can be attached to any annotation.
  - Notes are private and only visible to the user who created them.
  - Notes are accessible through a user's dashboard and within the annotation sidebar.

9.3 User Library & Organization

- **Library View**
  - A persistent "Library" page is accessible via the user's dashboard.
  - Displays:
    - Recently viewed documents
    - Favorite documents
    - Saved annotation bookmarks
    - Saved search queries
- **Search History**
  - Tracks documents recently edited or searched by the user.

- **Personal Organization**
  - Personal tags act as the main method of user-driven categorization.
  - Users can filter and group documents based on:
    - Language
    - Tier type
    - Media presence
    - Personal tags

9.4 Dashboard

- Upon login, users are shown a customizable dashboard containing:
  - Recently viewed documents
  - Favorite tags (most used personal/global tags)
  - Saved search queries
  - Progress indicators (e.g., % of annotations viewed, % reviewed in a document)

Section 10: Sharing, Groups, Permissions

10.1 Group & Collection Model

- In **public mode**, user groups (called "collections") are optional but available to support collaborative teams.
- In **non-public mode**, all documents must belong to a collection (i.e. a permissioned group). No access (view or edit) is granted outside group-based permissions.
- A collection is a Git repository, and also a permission boundary.
- Roles per collection:
  - **Viewer** (read-only, can bookmark, favorite, create personal tags)
  - **Contributor** (can create/edit their own documents within the collection)
  - **Moderator** (can manage/edit all documents within the collection)
  - **Admin** (instance-wide power, all collections and settings)
- Documents within a group can be subdivided further using the document group folder logic (see Section 2), but permissions are still controlled at the collection level.

10.2 Sharing & Invitations

- **Public documents** (in public mode) can be:
  - Shared via open link (read-only)
  - Commented on by logged-in users (if allowed by document owner)
  - Collaborated on in draft/pre-publish stage by invite
- **Restricted documents** (non-public mode or unpublished public docs):
  - Can only be accessed by group members with appropriate roles
  - Can be shared via invitation:
    - Invite by email (if no account, pre-approval required)
    - Invitation grants access after account creation and acceptance

    - Permission level (view, comment, edit) is set at the time of invite

- **Commenting**:
    - Can be enabled for specific invited users even after publishing
    - User-level permissions include: View Only / Comment / Edit

10.3 Embedding Options

- **Only available in public mode**
- Supported embed types:
    - Mini viewer for one or more annotations
    - Book-style layout of one double-page (e.g., bilingual)
    - Optional inclusion of media player for those snippets
- Embeds are read-only
- Users can choose whether their published document is embeddable

10.4 Finalization & Versioning

- Documents are **frozen** when published
    - No edits allowed to a published version
    - Further edits create a new version (with its own version ID)
- Each version has:
    - A UUID for citation and linking
    - (Optional) user-specified permanent ID (e.g. DOI, catalog string)
- Shared links to versions remain valid forever
- Web URLs may use either UUID or custom slug, depending on instance settings
- A document may be:
    - **In progress**
    - **Under review**
    - **Rejected**
    - **Published**

Section 11: Desktop Application

This section defines requirements for a standalone desktop version of the viewer and annotation platform, enabling full offline editing and review capabilities.

11.1 Purpose and Use Case

The desktop version is intended for:

- Users with limited or no internet access
- Secure editing environments (e.g. pre-publication sensitive data)
- Longform offline work or travel-based usage

It should mirror the functionality of the web version wherever possible, with some constraints outlined below.

11.2 Deployment Model

- Built using Electron or equivalent cross-platform wrapper
- Distributed as native installers for Windows, macOS, and Linux
- Auto-update support is desirable, with fallback to manual download
- Installation should include an embedded Git backend (e.g., bundled git binary or NodeGit)

11.3 Offline Capabilities

The app should function fully offline, supporting:

- Opening and browsing local repositories
- Media playback (audio/video files stored locally)
- Editing and creating:
    - Annotations
    - Inline spans and tags
    - Tier structures
    - Metadata
- Real-time collaboration is not required in offline mode
- Local commits saved directly to Git history
- Search, filters, and local tag queries operate against local manifests and indexes

11.4 Git Integration

Desktop app must allow:

- Cloning remote Git repositories
- Pushing/pulling to/from remote (e.g., GitHub, institutional Git servers)
- Support for SSH and HTTPS authentication
- Display commit history and allow revert/reset operations via UI
- Use Git LFS for large media files, consistent with web deployment
- Allow configuration of origin remotes, especially important for public vs non-public deployments

11.5 Conflict Resolution

On reconnect or sync:

- Detect changes between local and remote
- Attempt fast-forward merge automatically
- If conflict exists, present:
    - Visual diff viewer
    - Option to manually merge or discard local/remote changes

11.6 Document Syncing

- Repositories managed per-collection
- Collection-level settings (e.g., naming conventions, tagsets) are respected in desktop app
- On first clone, manifests and indexes are regenerated if missing
- Ability to "refresh" folder structure from Git HEAD manually

11.7 Optional Integration with Web App

If a user logs into the desktop app using the same credentials as on the web platform:

- Allow sync of dashboard state (e.g., recent files, resume points, favorites)
- Optional: configure hybrid mode, where desktop editing saves to local cache and pushes to server when online

Section 12: Localization

This section defines the system's ability to support multilingual user interfaces and user preferences, ensuring accessibility and usability for an international user base.

12.1 Internationalization (i18n)

- The entire application frontend must be built with full i18n support.
- All user-facing UI strings must be extracted to translation files (e.g., JSON or PO format).
- Language files can be loaded dynamically based on user or system settings.

12.2 Default Language

- Each deployed app instance must have a system-wide default language.
- This is configured at install/deploy time but may be updated later by an admin user.

12.3 User-Specific Language Preferences

- Authenticated users may select their own interface language from the list of installed translations.
- This preference is stored per user account and persists across sessions.

12.4 Language Availability

- Admin users can manage available interface languages:
  - Enable or disable available translations
  - Upload custom translation files
  - Track completeness status for partial translations

12.5 Scope of Localization

- The following UI components must be fully translatable:
  - Navigation labels
  - Button and form labels
  - Status messages
  - Dialogs, tooltips, and modals
  - Dashboard, Library, and Settings interfaces
  - Error and fallback states
- User-generated content (document titles, tags, annotation text) is not translated by the system but may contain multilingual data.

12.6 Fallback Behavior

- If a selected language file is incomplete or missing a string:
    - Fallback to the system default language
    - Missing strings should be logged to the console or error tracking

Section 13: System-Wide Settings

13.1 Instance Configuration

Each deployment of the app must define whether it operates in:

- **Public mode**: open-access model with optional login, publishing workflow.
- **Non-public mode**: strict permissions via group-based access; all documents belong to a collection.

This setting is:

- **Hardcoded at deploy time.**
- **Not changeable via the admin UI post-deployment** (requires full export/import for migration).

13.2 Naming Conventions

Each app instance may define:

- A default naming convention (e.g., COLL-001, BNDL-XYZ-0002).
- A selectable list of naming templates usable when creating new documents or folders.
- Admins may create new naming templates via UI.

Per-Collection Overrides

- Contributors may apply collection-specific naming templates.
- These influence how new folders or bundles are generated.

13.3 Tier Type Configuration

- Tier types are defined at the **collection level** but can follow instance-wide vocabularies.
- Supports:
    - UI-based creation of new tier types and their relationships.
    - ELAN type mappings preserved for import/export.
- Admins may define a global vocabulary template for collections to inherit from.

13.4 Tagset Management

- Tagsets are managed per collection, stored in Git alongside other collection data.

Admins may define:

- A default global tagset template (optional).
- Preset tagsets for users to select during new collection setup.

Tagsets:

- Are versioned via Git.

- May include category hierarchies.
- Editable in UI via tag manager interface.

13.5 Publishing Workflows

- Document states (e.g., draft, in review, published) are defined instance-wide.
  - This configuration is stored in system config files, **not editable from UI**.
- Citation templates (e.g., BibTeX) are managed by admins.

Visibility Defaults

- Controlled by the deploy mode (public vs non-public), not per-document.

13.6 Feature Toggles

Configurable UI behaviors include:

- Dashboard visibility
- Resume-state tracking
- Glossary panel toggle
- Media player layout (compact/full)
- Max simultaneous pages visible

These are:

- Instance-level settings
- Editable from the system config or admin UI